**Exercise 1: Configuring a Basic Spring Application**

**Scenario:**

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

**Steps:**

1. **Set Up a Spring Project:**
   * Create a Maven project named **LibraryManagement**.
   * Add Spring Core dependencies in the **pom.xml** file.

**Pom.xml:**

<dependencies>

<!-- Spring Core -->

<dependency>

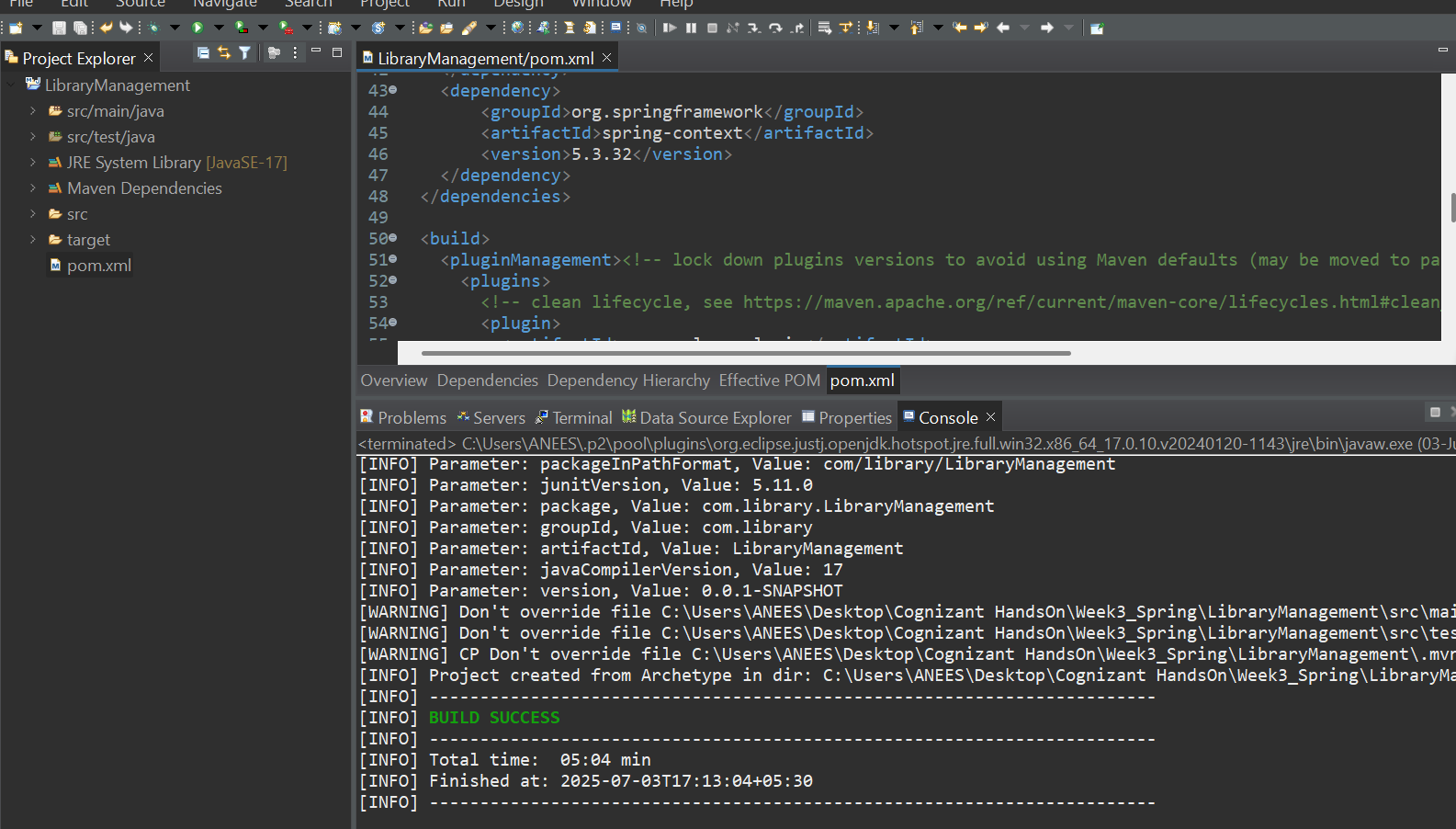
<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>



2. **Configure the Application Context:**

* + Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
  + Define beans for **BookService** and **BookRepository** in the XML file.

***applicationContext.xml*** *:*

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Bean for BookRepository -->

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<!-- Bean for BookService -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**3.Define Service and Repository Classes:**

1. Create a package **com.library.service** and add a class **BookService**.
2. Create a package **com.library.repository** and add a class **BookRepository**.

***BookRepository.java***

package com.library.repository;

public class BookRepository {

public void save(String bookName) {

System.***out***.println("Book '" + bookName + "' saved to the repository.");

}

}

***BookService.java***

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.out.println("Adding book: " + bookName);

bookRepository.save(bookName);

}

}

**4.Run the Application:**

Create a main class to load the Spring context and test the configuration.

***MainApp.java***

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml");

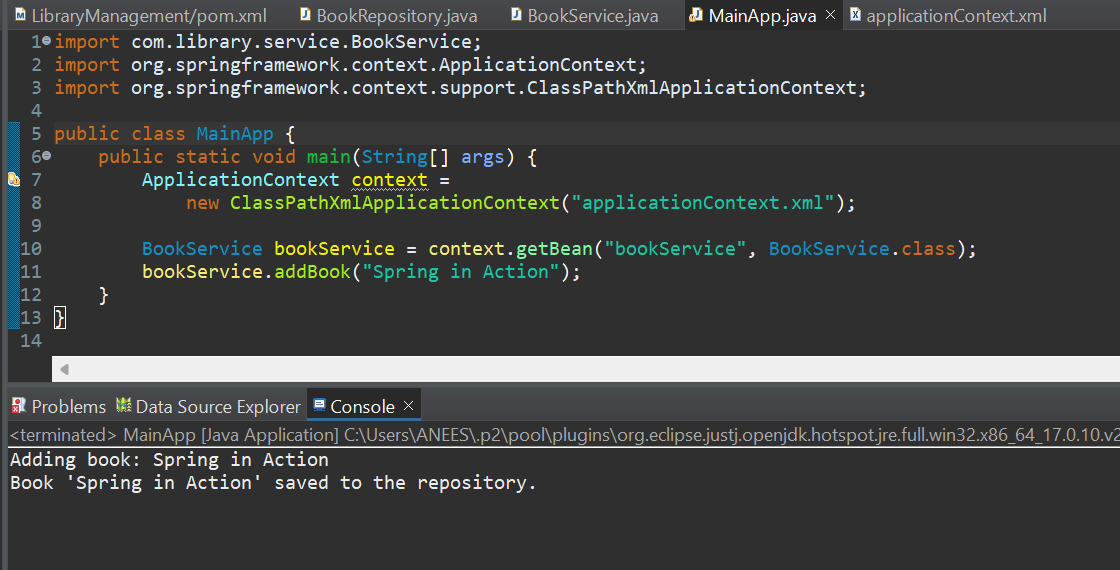
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Spring in Action");

}

}

**Output:**



**Exercise 2: Implementing Dependency Injection**

**Scenario:**

In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

**Steps:**

1. **Modify the XML Configuration:**
   * Update **applicationContext.xml** to wire **BookRepository** into **BookService**.

***applicationContext.xml***

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Step 1: Define the BookRepository Bean -->

<bean id="bookRepository" class="com.example.repository.BookRepository"/>

<!-- Step 1: Define the BookService Bean and inject BookRepository -->

<bean id="bookService" class="com.example.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

1. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.

***BookRepository.java***

package com.example.repository;

public class BookRepository {

public void save() {

System.out.println("Saving book into the repository...");

}

}

***BookService .java***

package com.example.service;

import com.example.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void performService() {

System.***out***.println("BookService is working...");

bookRepository.save();

}

}

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

***LibraryManagementApplication.java***

package com.example.librarymanagement;

import com.example.librarymanagement.service.BookService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication implements CommandLineRunner {

@Autowired

private BookService bookService;

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

@Override

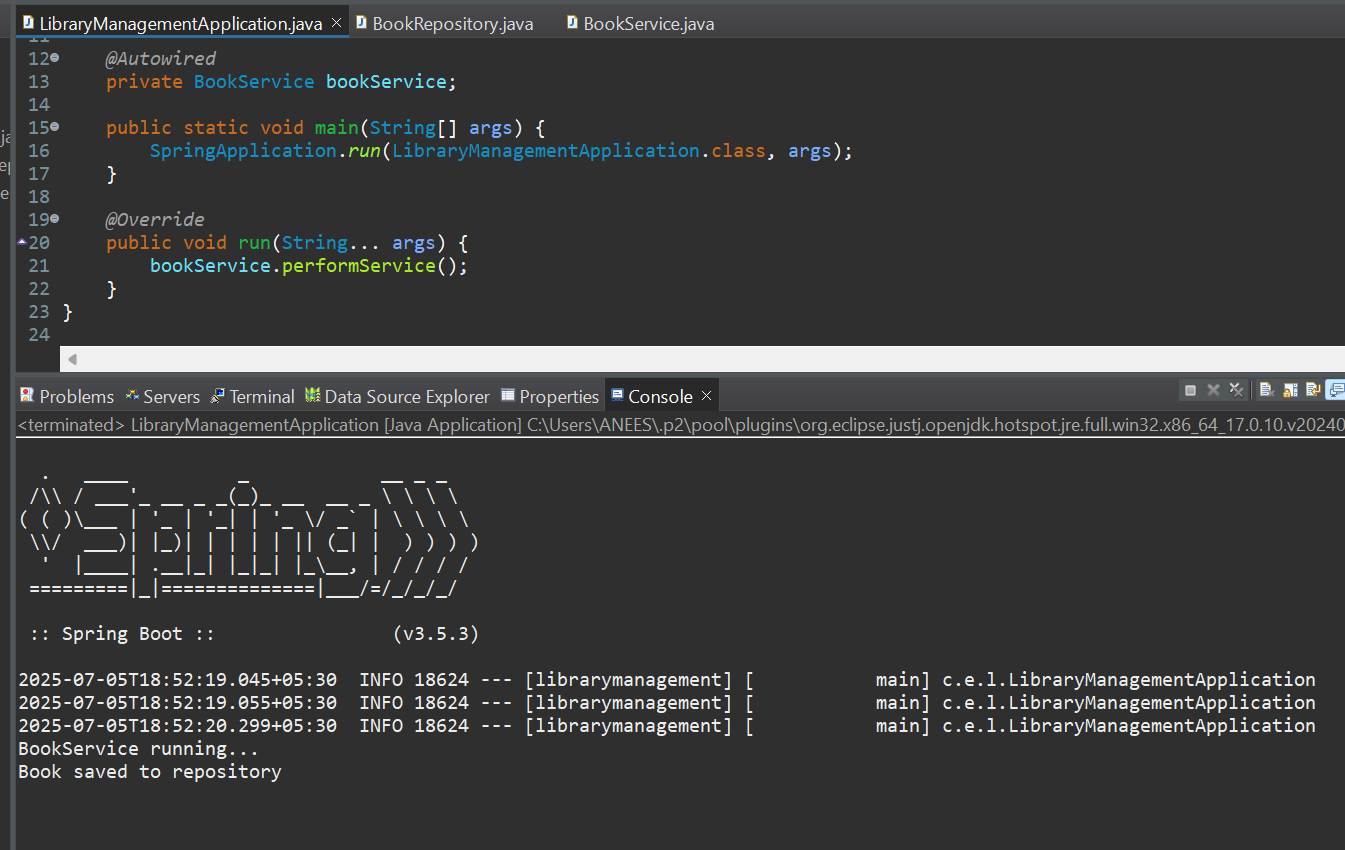
public void run(String... args) {

bookService.performService();

}

}

**Output:**



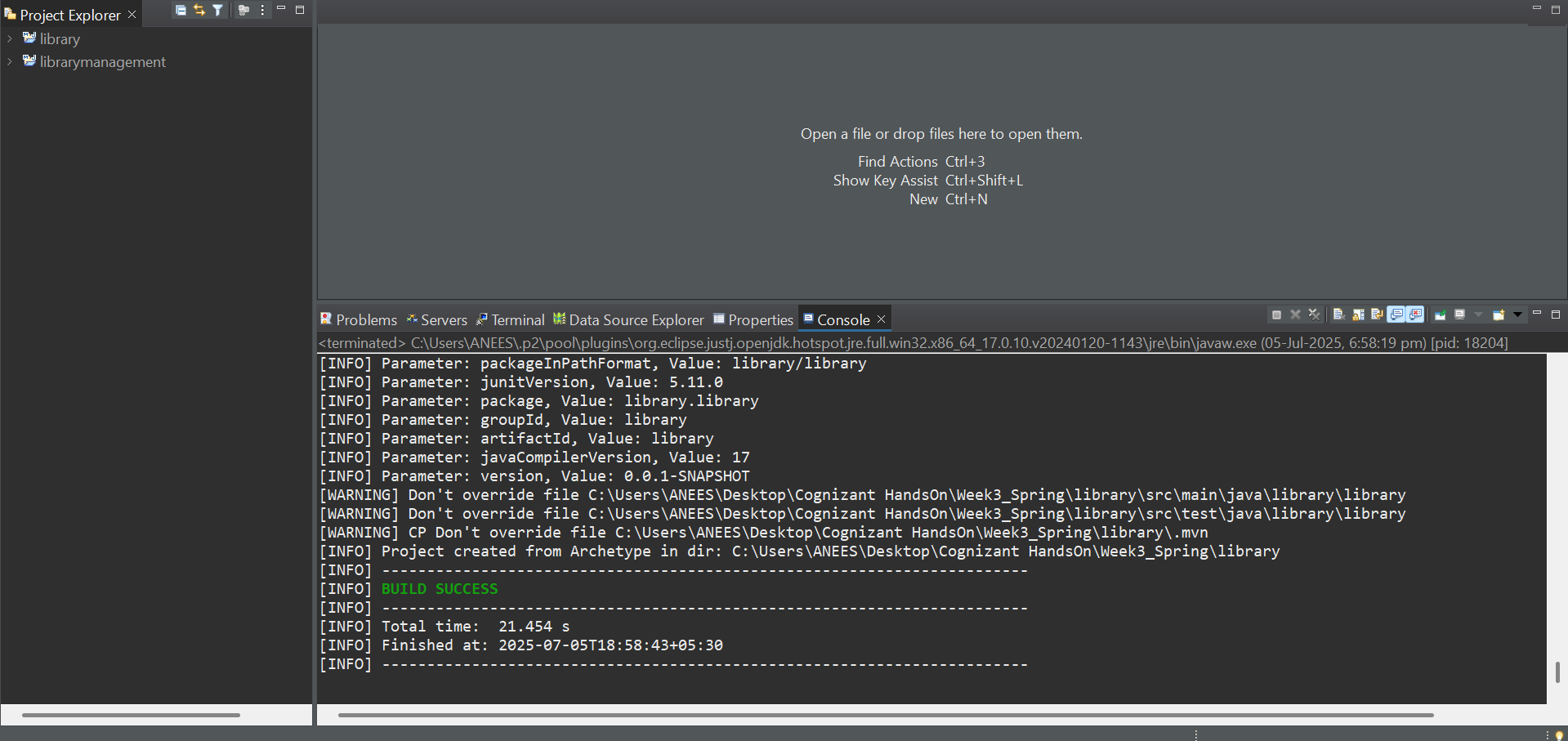
**Exercise 4: Creating and Configuring a Maven Project**

**Scenario:**

You need to set up a new Maven project for the library management application and add Spring dependencies.

1. **Create a New Maven Project:**

Create a new Maven project named **LibraryManagement**.



**2.Add Spring Dependencies in pom.xml:**

* + Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

***Pom.xml:***

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0.0</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Spring Context for DI -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring AOP (Aspect-Oriented Programming) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Logging support -->

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin for Java 1.8 -->

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.11.0</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**3.Run the Application:**

***MainApp.java:***

package com.librarymanagement;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

System.out.println("Library Management Application Started");

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

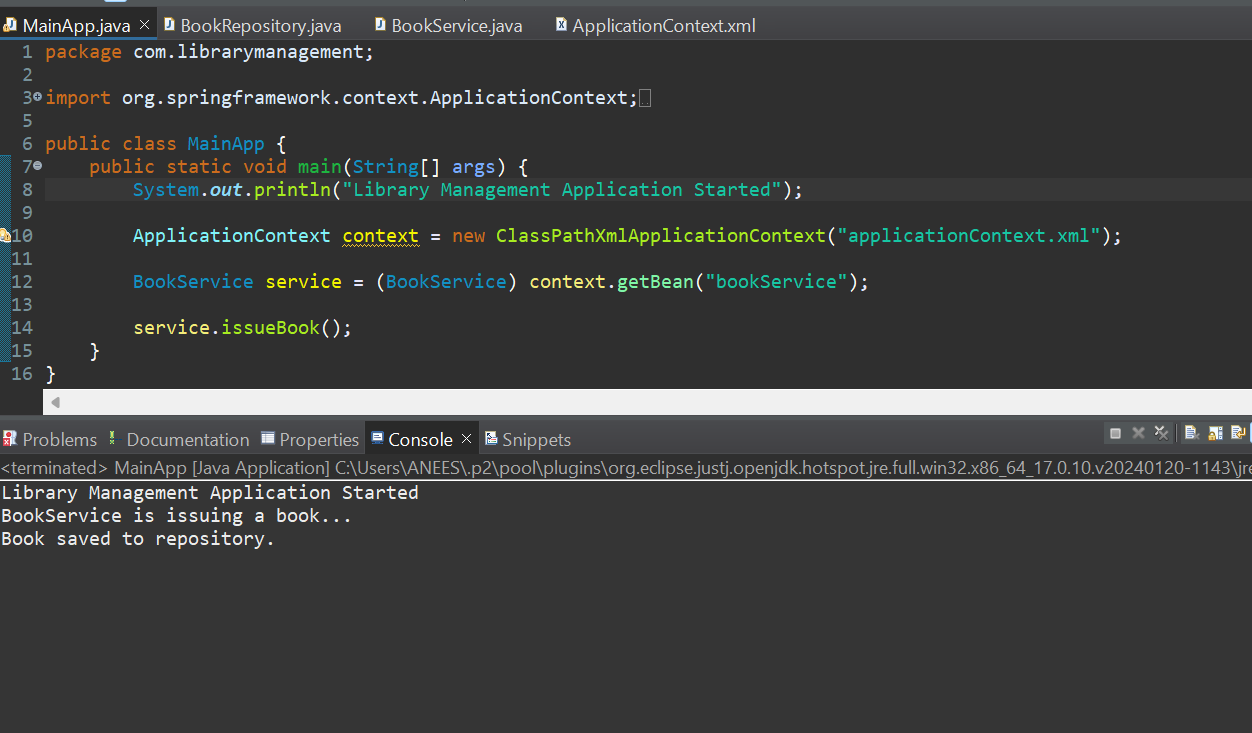
BookService service = (BookService) context.getBean("bookService");

service.issueBook();

}

}

**Output:**



**Exercise 5: Configuring the Spring IoC Container**

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**Steps:**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.

***applicationContext.xml*** *:*

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bookRepository" class="com.librarymanagement.BookRepository" />

<bean id="bookService" class="com.librarymanagement.BookService">

<property name="bookRepository" ref="bookRepository" />

</bean>

</beans>

1. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.

***BookService.java***

package com.librarymanagement;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void display() {

bookRepository.print();

}

}

***BookRepository.java***

package com.librarymanagement;

public class BookRepository {

public void print() {

System.out.println("BookRepository: Repository is active.");

}

}

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

***MainApp.java***

package com.librarymanagement;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

System.out.println("Library Management Application Started");

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

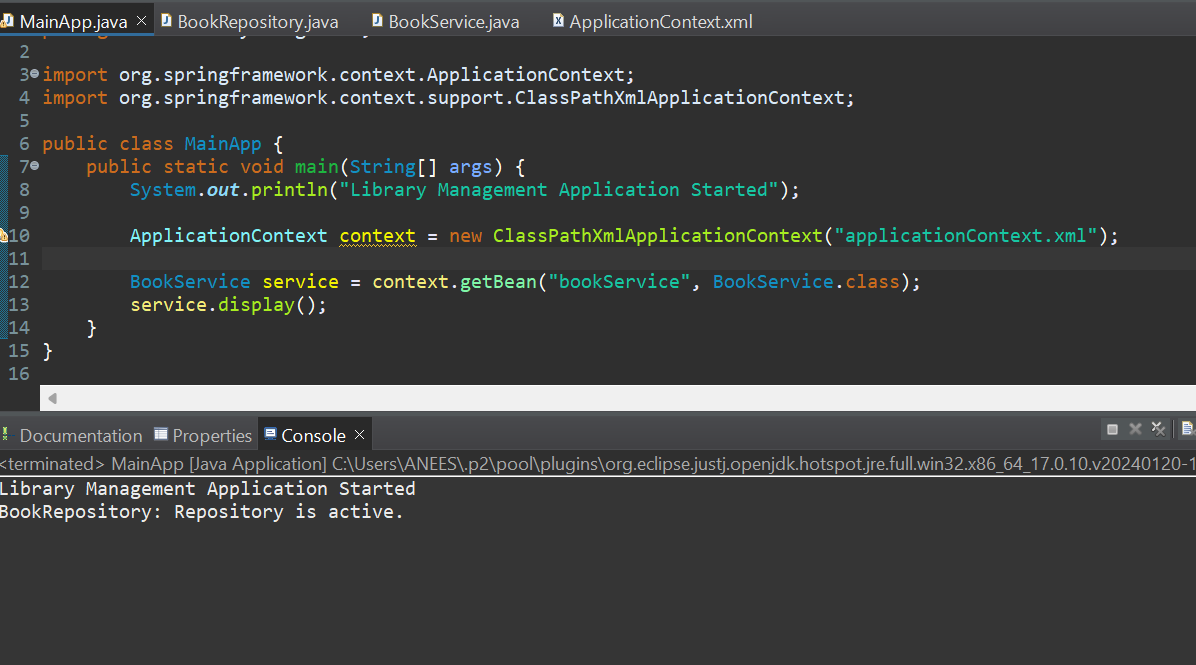
BookService service = context.getBean("bookService", BookService.class);

service.display();

}

}

**Output**



**Exercise 7: Implementing Constructor and Setter Injection**

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Steps:**

1. **Configure Constructor Injection:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Bean for BookRepository -->

<bean id="bookRepository" class="com.librarymanagement.BookRepository" />

<!-- Constructor and Setter Injection for BookService -->

<bean id="bookService" class="com.librarymanagement.BookService">

<!-- Constructor injection -->

<constructor-arg value="Library Book Issuer Service" />

<!-- Setter injection -->

<property name="bookRepository" ref="bookRepository" />

</bean>

</beans>

1. **Configure Setter Injection:**
   * Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.

***BookRepository .java***

package com.librarymanagement;

public class BookRepository {

public void print() {

System.out.println("BookRepository is working.");

}

}

***BookService.java***

package com.librarymanagement;

public class BookService {

private String serviceName;

private BookRepository bookRepository;

// Constructor for constructor injection

public BookService(String serviceName) {

this.serviceName = serviceName;

System.out.println("Constructor Injection: serviceName = " + serviceName);

}

// Setter for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Setter Injection: bookRepository injected");

}

public void display() {

System.out.println("BookService: " + serviceName);

bookRepository.print();

}

}

1. **Test the Injection:**
   * Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

***MainApp.java***

package com.librarymanagement;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

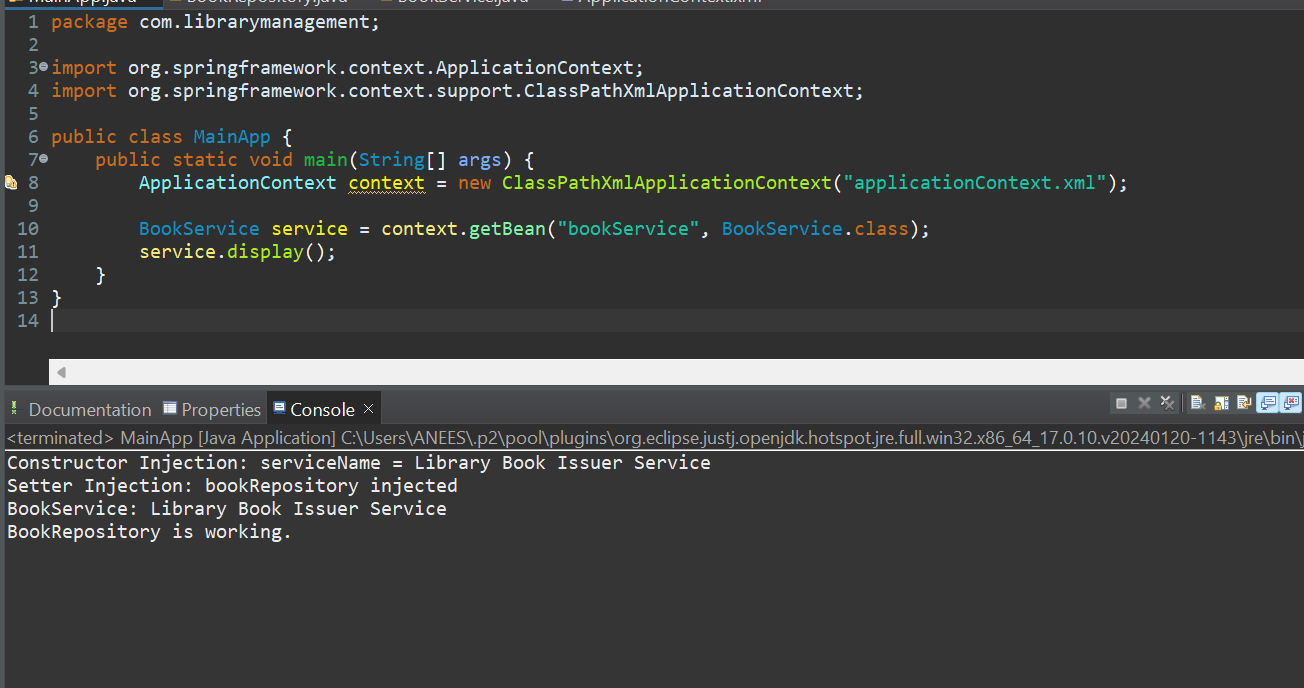
BookService service = context.getBean("bookService", BookService.class);

service.display();

}

}

**Output**



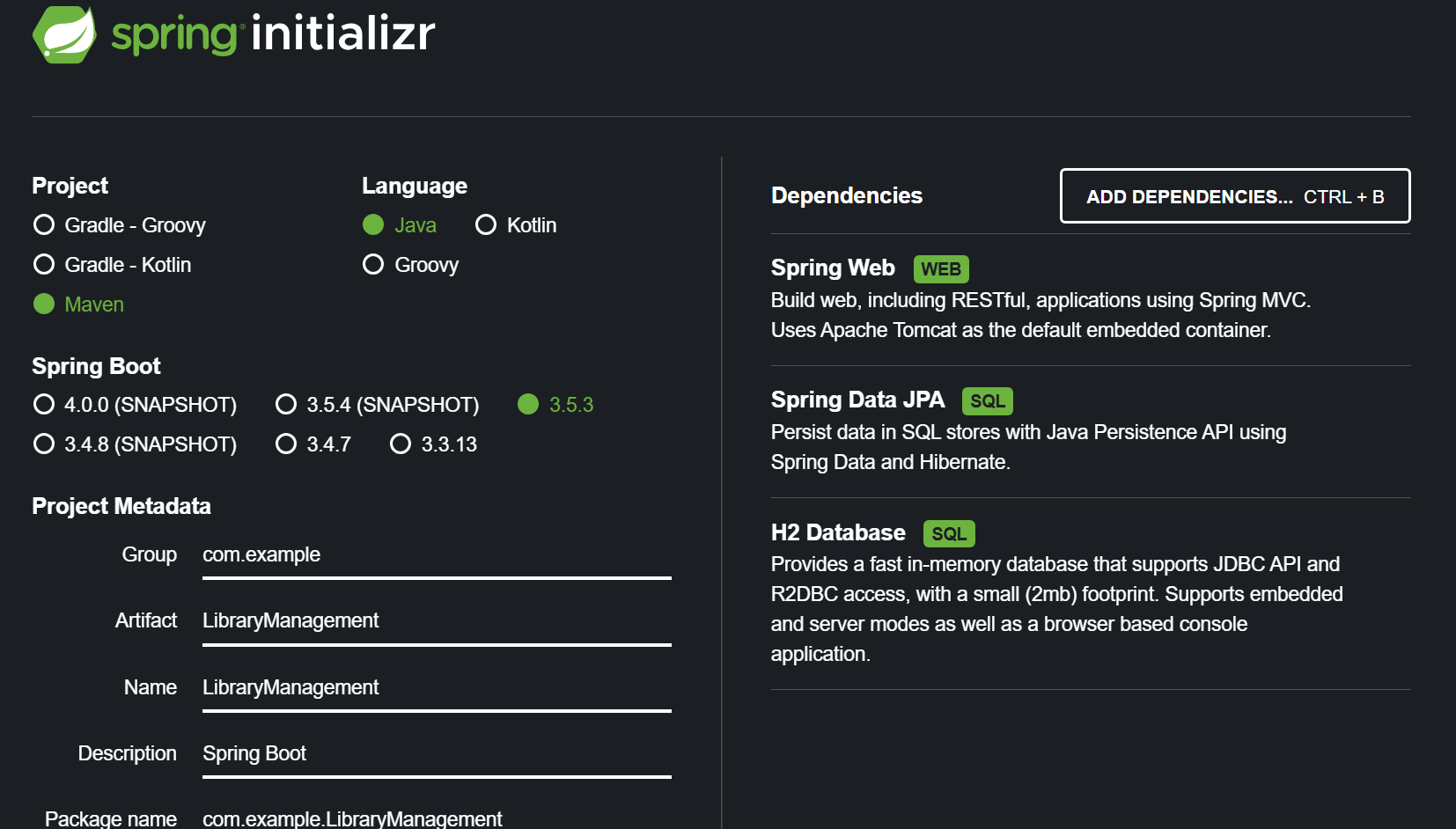
**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

1. **Create a Spring Boot Project:**
   * Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.



1. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.

***Dependencies in Pom.xml***

<dependencies>

<!-- Spring Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

1. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.

***application.properties***

spring.application.name=LibraryManagement

# H2 in-memory DB config

spring.datasource.url=jdbc:h2:mem:librarydb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# JPA config

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Enable H2 Console

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

1. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.

***Book.java***

package com.example.LibraryManagement.model;

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

public Book() {}

public Book(String title, String author) {

this.title = title;

this.author = author;

}

// Getters and setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getTitle() { return title; }

public void setTitle(String title) { this.title = title; }

public String getAuthor() { return author; }

public void setAuthor(String author) { this.author = author; }

}

***BookRepository .java***

package com.example.LibraryManagement.repository;

import com.example.LibraryManagement.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

1. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.

***BookController.java***

package com.example.LibraryManagement.controller;

import com.example.LibraryManagement.model.Book;

import com.example.LibraryManagement.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return bookRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {

Book book = bookRepository.findById(id).orElse(null);

if (book != null) {

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

return bookRepository.save(book);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

1. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

***LibraryManagementApplication.java***

package com.example.LibraryManagement;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication {

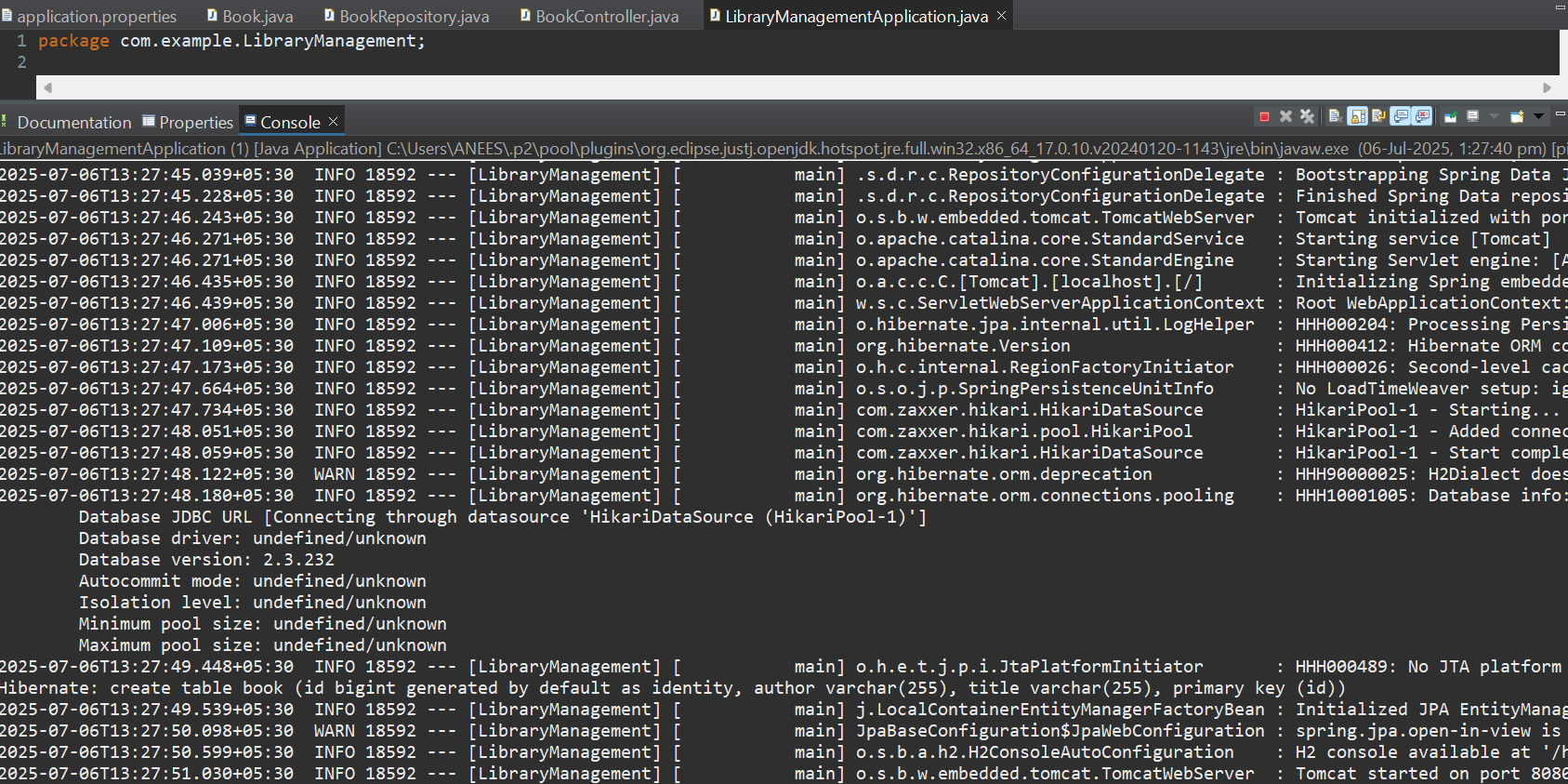
public static void main(String[] args) {

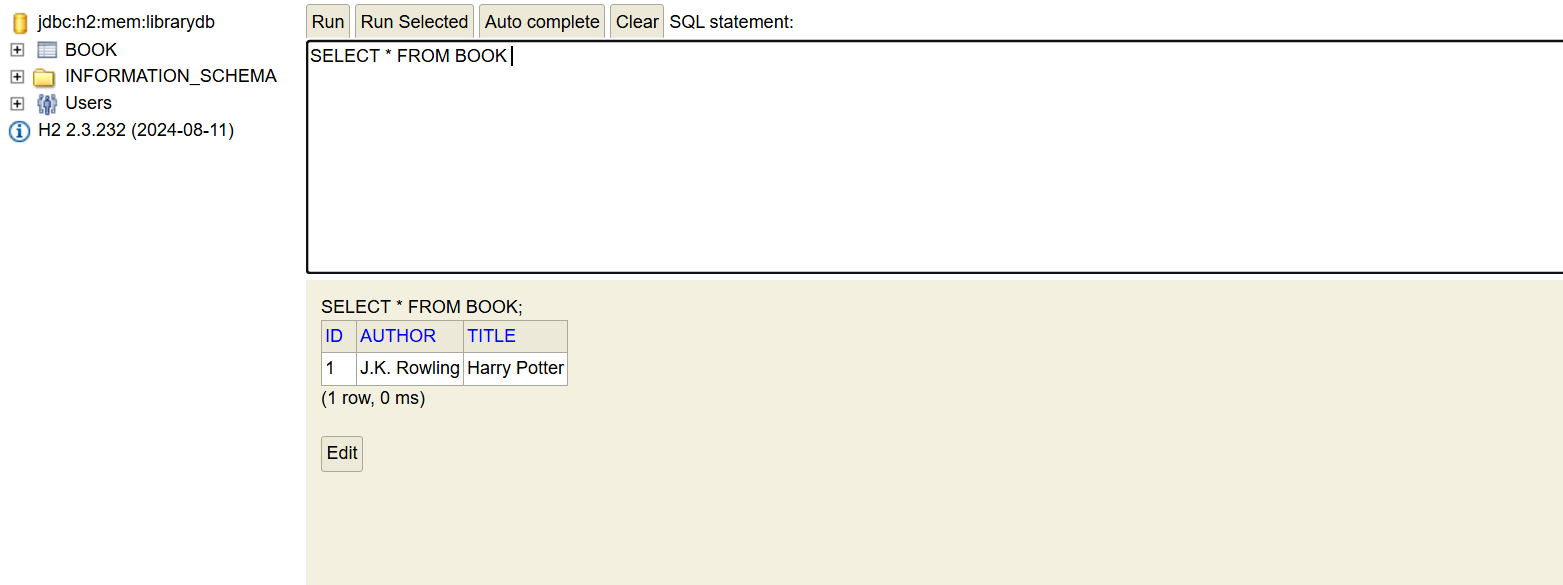
SpringApplication.run(LibraryManagementApplication.class, args);

}

}

**Output**

****

****